1 Objectives

By the end of this lecture, you should

- Be able to explain how large data structures in a functional language differ from similar structures in an imperative environment
- know how to simulate updates in a persistent environment
- be able to add undo functionality using persistent data

2 Announcements

3 Examples

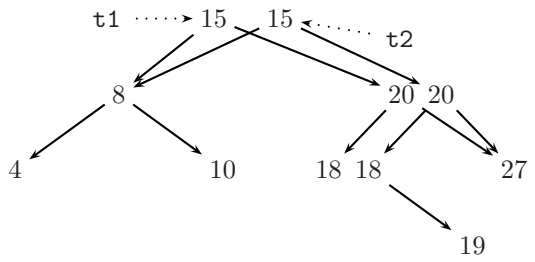
The Interpreter

```
let eval commands =
 let rec aux tree commands stack =
  match commands with
    (Add i)::xs ->
            aux (add i tree) xs (tree::stack)
  | Show :: xs ->
            show tree ; aux tree xs stack
  | Undo :: xs ->
            aux (List.hd stack) xs (List.tl stack)
  | [] -> print_string "Done\n" in
  aux Empty commands []
```

4 Problems

Try the following problems. In a few minutes the instructor will go over the solutions. Feel free to work with the person next to you!

1. In the lecture we discussed the consequences of adding a second new element to the tree. Draw the result of the command `(define t3 (add t2 9))`.



2. Write the `delzeros` function, but make it able to share the data.